PATENT APPLICATION
Attorney Docket No. 2705-128

COMBINED DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled FULLY DISTRIBUTED, SCALABLE INTERFACE, COMMUNICATION SYSTEM, the specification of which:

[X] is attached hereto.
[X] was filed on August 8, 2000 as Application No. 60/223,824
[ ] and was amended on ______________(if applicable)
[ ] with amendments through ____________ (if applicable).

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the patentability of this application in accordance with Title 37, Code of Federal Regulations, Sec. 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, Sec. 119 (a)-(d) or §365(b) of any foreign application(s) for patent or inventor's certificate, or §365(a) of any PCT international application which designated at least one country other than the United States of America, listed below and have also identified below any foreign application for patent or inventor's certificate, or of any PCT international application having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)			Claiming Priority?	
____________	____________________	____________________	[ ]	[ ]
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No

I hereby claim the benefit under Title 35, United States Code, Sec. 119(e) of any United States provisional application listed below:

Provisional Application No.	Filing Date
60/223,824	August 8, 2000

I hereby claim the benefit under Title 35, United States Code, Sec. 120 or §365(c) of any PCT international application designating the United States of America listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Sec. 112, I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, Sec. 1.56 which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

(Application No.)	(Filing Date)	(Status) (patented, pending, abandoned)

I hereby appoint the following attorneys to prosecute the application, to file a corresponding international application, to prosecute and transact all business in the Patent and Trademark Office connected therewith:

Customer No. 20575

Attorney Name	Registration No.
Jerome S. Marger	26,480
Alexander C. Johnson, Jr.	29,396
Alan T. McCollom	28,881
James G. Stewart	32,496
Glenn C. Brown	34,555
Stephen S. Ford	35,139
Julie L. Reed	35,349
Gregory T. Kavounas	37,862
Scott A. Schaffer	38,610
Joseph S. Makuch	39,286
James E. Harris	40,013
Graciela G. Cowger	42,444
Ariel Rogson	43,054
Craig R. Rogers	43,888

Direct all telephone calls to Julie L. Reed at (503) 222-3613 and send all correspondence to:

Julie L. Reed
Marger Johnson & McCollom, P.C.
1030 SW Morrison Street
Portland, OR 97205

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of first inventor:	Peter Michael Gits	
Inventor's signature:	[illegible]	9/26/2000 (Date)
Residence:	Agoura Hills, California	
Citizenship:	United States	
Post Office address:	5550 Fairgrange Drive Agoura Hills, California 91301	
Full name of second joint inventor:	Dale J. Seavey	
Inventor's signature:	[illegible]	9·22·2k (Date)
Residence:	Sunol, California	
Citizenship:	United States	
Post Office address:	1180 Kilkare Road Sunol, California 94586	

09675474.092600

Appendix

```
* Title:        Sip Test<p>
package JiniDiscussion.SipTest;
import java.rmi.*;
import net.jini.space.JavaSpace;
import net.jini.core.lease.*;
import net.jini.core.transaction.*;
import net.jini.core.entry.UnusableEntryException;
import java.util.Vector;
import JiniDiscussion.SipTest.*;
import net.jini.admin.Administrable;
import com.sun.jini.outrigger.*;
import net.jini.core.entry.*;
import net.jini.core.event.*;
import java.rmi.server.*;
import java.rmi.RemoteException;
import JiniDiscussion.SipTest.SpaceUtils;


    public class DoubleAgentThread extends Thread implements
RemoteEventListener
    {
        EventRegistration eventRegSip            = null;
        EventRegistration eventTraceRoute = null;
        JavaSpace StartSpace                     = null;
        public DoubleAgentEntry daEntry          = null;
        static int      iMaxLeaseFor                   = 1000 * 60 * 1;
        boolean bThreadStillGoing                = false;
      static int iLeaseFor                       = 1000*60*1;
      static int iMarginOfError                  = 1000*4;
        SpaceUtils      spUtil                               = null;
        SipEntry sipOriginalEntry                = null;
        boolean bCallActive                            = true;
            SipEntry seSnapTemplate                    = null;
        long lLastTimeStamp                            = 0L;
        Entry snapImpl                                 = null;

        DoubleAgentSpaceListener SystemAudit= null;
      boolean bDeleteMark                  = false;
      long lDeleteSanctioned               = 0L;
String strDestinationPort          = null;
        String strDestinationAddress        = null;
        RegistrationEntry tReg              = null;
        SipUdpServer sipUdpServer           = null;
        static long lWaitOnTransFor         = 2L * 1000L;
        boolean bLeaseCancelled             = false;

      public DoubleAgentThread(DoubleAgentSpaceListener parent, JavaSpace
tSpace, DoubleAgentEntry daEntry){
            this.SystemAudit                  = parent;
            this.StartSpace                        = tSpace;
            this.daEntry                           = daEntry;
                  this.sipOriginalEntry            = new SipEntry();
            this.sipOriginalEntry.Copy(daEntry.sipCopyEntry);
            this.tReg                       = daEntry.RegistrationEntryFor;


if(daEntry.sipCopyEntry.WhereThisMessageIsFrom.compareTo(this.tReg.strDevic
eIpAddress) != 0){
                System.out.println("Where the message came from doesn't
match the registration, ignoring the registration");
                System.out.println("this.tReg.strDeviceIpAddress = " +
this.tReg.strDeviceIpAddress);
```

```
System.out.println("daEntry.sipCopyEntry.WhereThisMessageIsFrom = " +
daEntry.sipCopyEntry.WhereThisMessageIsFrom);
                this.tReg.strDeviceIpAddress =
daEntry.sipCopyEntry.WhereThisMessageIsFrom;
            }
            }

        public void openServerSocket(){
            try {
                this.sipUdpServer  = new SipUdpServer();
                this.sipUdpServer.CreateUdpConnection(0);
            }catch(Exception ee){
                ee.printStackTrace();
            }
        }

        public void run(){
            try {
            UnicastRemoteObject.exportObject(this);
            }catch(RemoteException re){
                  re.printStackTrace();
                return;
            }
            openServerSocket();
            if(this.daEntry.bDeliverEmbeddedEntry != null)
                if(this.daEntry.bDeliverEmbeddedEntry.booleanValue() ==
true)
                    deliverEmbeddedEntry(this.daEntry);
            SipEntry sippy                = new SipEntry();
            sippy.ToIndividual            = this.daEntry.strPickupFor;
            sippy.DiscussionId            = this.daEntry.strCallId;
            sippy.bForwardAlert           = new Boolean(false);
            sippy.bSmartAlert       = new Boolean(false);
            try {
                  eventRegSip             = this.StartSpace.notify(sippy,
null, this, this.iMaxLeaseFor, null);
            }catch(RemoteException re){
                  re.printStackTrace();
            }catch(TransactionException te){
                  te.printStackTrace();
            }
            deliverEntry();
                  keepRegistrationsAlive();
            System.out.println("terminating notification lease:
DoubleAgent for Conference id = " + this.daEntry.strCallId);
            this.destroy();
        }

      public void keepRegistrationsAlive(){
            boolean bInterrupted     = false;
            long lnowTime                 = System.currentTimeMillis();
            long lSipLeaseExpires    = lnowTime;
            long ldiff                  = 0l;
            int iInternalCounter     = 0;
            if(eventRegSip != null){
            try {
                  lSipLeaseExpires   =
eventRegSip.getLease().getExpiration();
            }catch(Exception le){
                  le.printStackTrace();
                  return;
            }
            while(bInterrupted == false && this.bCallActive == true)
            {
                  try{
                        if((lSipLeaseExpires - lnowTime < this.iLeaseFor +
this.iMarginOfError) && this.bCallActive == true)
                    {
                        try {
                              eventRegSip.getLease().renew(this.iLeaseFor);
                              lSipLeaseExpires  =
eventRegSip.getLease().getExpiration();
                        }catch(Exception le){
                              le.printStackTrace();
                              return;
                        }
                        iInternalCounter++;
                        if(iInternalCounter % 10 == 0){
                            iInternalCounter = 0;
                            System.out.println("DoubleAgentThread = " +
this.daEntry.strPickupFor);
                        }
                        else
                            System.out.print("DoubleAgentThread = " +
this.daEntry.strPickupFor);
                    }
                    if(this.bCallActive == false)
                        break;
                    else
                              sleep(this.iLeaseFor/3);
                    lnowTime  = System.currentTimeMillis();
                 }catch(InterruptedException ie){
                        ie.printStackTrace();
                 bInterrupted = true;
                 break;
                 }
            }
            bLeaseCancelled = true;
            System.out.println("Call completed, cancelling
DoubleAgentThread lease");
            try {
                eventRegSip.getLease().cancel();
            }catch(UnknownLeaseException ule){  ule.printStackTrace();
            }catch(RemoteException re){               re.printStackTrace();
            }
      }
         }

            public void destroy(){
            System.out.println("removing notification lease for
DoubleAgentThread."+ this.daEntry.strPickupFor);
            if(eventRegSip != null && bLeaseCancelled == false)
                  try{
                        eventRegSip.getLease().cancel();
                  }catch(UnknownLeaseException ule){
      ule.printStackTrace();
                  }catch(RemoteException re){
      re.printStackTrace();
                  }
          if(this.sipUdpServer != null)
                if(this.sipUdpServer.UdpSocket != null){
                    try {
                        sipUdpServer.UdpSocket.close();
                        sipUdpServer.UdpSocket.disconnect();
                      }
                        catch(Exception sockErr){
                        sockErr.printStackTrace();
                              System.out.println(sockErr.getMessage());
                  }
                }
                this.sipUdpServer = null;
            }

        public void notify(RemoteEvent theEvent) throws
  UnknownEventException, RemoteException
        {

   "BetweenerThread.notify()", "notify", theEvent, true);
              if(theEvent.getID() == this.eventRegSip.getID() &&
  theEvent.getSource().equals(this.eventRegSip.getSource()) ){
                    System.out.println("DoubleAgentThread.notify()");
                    deliverEntry();
              }
        }

          public void deliverEntry(){
              SipEntry readSip              = null;
              boolean bContinueChecking     = true;

              seSnapTemplate                               = new SipEntry();
              seSnapTemplate.ToIndividual   = this.daEntry.strPickupFor;
              seSnapTemplate.DiscussionId   = this.daEntry.strCallId;
              seSnapTemplate.bForwardAlert  = new Boolean(false);
              seSnapTemplate.bSmartAlert         = new Boolean(false);
              while(bContinueChecking){
                    try {
                    readSip = (SipEntry)
  this.StartSpace.takeIfExists(seSnapTemplate, null, lWaitOnTransFor);
                          if(readSip != null){
                          forwardToDestination(readSip);

        markThreadForFutureDeleteIfDone(readSip);
                          SystemAudit.setFilter(Thread.currentThread(),
  readSip);
                          SystemAudit.outPrintln("sent sipEntry ---> to " +
  readSip.ToIndividual + readSip.toString());
                          System.out.println("sent sipEntry ---> to " +
  readSip.ToIndividual + readSip.toString());
                       }
                       else {
                          bContinueChecking = false;
                       }
                          this.lLastTimeStamp     =
  System.currentTimeMillis();
                    }catch(RemoteException re){
        re.printStackTrace();
                    }catch(TransactionException te){    te.printStackTrace();
              }catch(InterruptedException ie){    ie.printStackTrace();
                    }catch(UnusableEntryException ue){  ue.printStackTrace();
                    }catch (Exception e){
        e.printStackTrace();
                    }
              }
          }


          public boolean deliverEmbeddedEntry(DoubleAgentEntry daEnt){
              SipEntry readSip    = daEnt.sipCopyEntry;
              try {
                    if(readSip != null){
                      this.lLastTimeStamp    = System.currentTimeMillis();
                      return(forwardToDestination(readSip));
                  }
              }catch (Exception e){                          e.printStackTrace();
              }
              return false;
          }
```

```
            public boolean forwardToDestination(SipEntry sipForward){
                if(sipUdpServer == null)
                    return false;
                else if(sipUdpServer.UdpSocket == null)
                    return false;
                return(sipUdpServer.sendThreadSpaceToSip(tReg, sipForward));
            }

            public void completeCall(){
                this.bCallActive = false;
            }

            public String getCallId(){
                return this.daEntry.strCallId;
            }

            public long getLastActivity(){
                return this.lLastTimeStamp;
            }

            public long setLastActivity(long tStamp){
                this.lLastTimeStamp = tStamp;
                return this.lLastTimeStamp;
            }
                public boolean isMarkedForDelete(){
                return this.bDeleteMark;
                }

          public long getDeleteMarkedTime(){
                return this.lDeleteSanctioned;
          }

          public void MarkDeleted(){
                this.bDeleteMark          = true;
                this.lDeleteSanctioned    = System.currentTimeMillis();
                this.completeCall();
          }

          public void markThreadForFutureDeleteIfDone(SipEntry sipMessage){
                if(this.bDeleteMark == true)
                      return;
                int iIndexOfCANCEL   = sipMessage.SipStuff.indexOf("CANCEL");
                int iIndexOfBYE      = sipMessage.SipStuff.indexOf("BYE");
                if( (sipMessage.SipStatus.compareTo("BUSY")      == 0 )  ||
                      (sipMessage.SipStatus.compareTo("Busy")      == 0 )   ||
                      (sipMessage.SipStatus.compareTo("BYE")       == 0 )||
                      (sipMessage.SipStatus.compareTo("OK")        == 0 &&
iIndexOfCANCEL > 0 ) ||
                      (sipMessage.SipStatus.compareTo("CANCEL")    == 0 &&
iIndexOfCANCEL > 0 ) ||
                      (sipMessage.SipStatus.compareTo("OK")        == 0 &&
iIndexOfBYE > 0 )    ||
                      (sipMessage.SipStatus.compareTo("ACK")       == 0 &&
iIndexOfBYE > 0 )
                ){
                      this.MarkDeleted();
                      System.out.println("\n\n\t\tMarking " +
this.daEntry.strPickupFor + " thread for Future Delete");
                }
          }
        }
```